Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 5 (2011) 296–303

Procedia Computer Science

The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011)

A Distributed Architecture for Micro Context-Aware Agents

Patrice Roy[a], Bessam Abdulrazak[a], Yacine Belala[b]

[a]*Université de Sherbrooke, 2500, boul. de l'Université, Sherbrooke (Québec) CANADA J1K 2R1*
[b]*INRS-EMT, 800, de la Gauchetière O., Montréal (Québec) CANADA H5A 1K6*

Abstract

Open intelligent space is a space where mobile Agents can move in an unconstrained physical manner. While such a space enables a wide range of contextual applications that address the needs of mobile users, it introduces a new set of issues due to the dynamic nature of the Agent network. To address these issues, we propose a distributed architecture for autonomic Agents that relies on a micro-level, local, approach to Context Awareness. The underlying Context and programming models are presented, including the mechanisms and strategies used for Agent life cycle management, network isolation and Context request, processing and publishing.

Keywords: Context, Context-awareness, middleware, autonomic computing, intelligent Agents.

1. Introduction

Smart spaces are physical locations where software components bridge the gap between the logical world of software and the physical world of humans. Where traditional smart spaces have physical boundaries, open intelligent space [1] is an unconstrained smart space for fully mobile Agents. The dynamic nature of the resulting Agent network introduces a new set of issues, for which autonomic Agents [2] are required and for which a micro Context-awareness approach is appropriate [3].

Smart spaces are a form of augmented reality, such as was first described by Weiser [4], composed of both data and metadata describing both software and the physical world, resulting in a very complex problem space. Agents in smart spaces manage this complexity by operating on organized subsets of the information available, deemed to be Context in agreement with the definition given by Winograd, i.e. “Context is an operational term: Something is context because of the way it is used in interpretation, not due to its inherent properties” [5]. Agents that consume, process or produce Context are said to be Context-dependent [6], or Context-aware if they “[use] context to provide relevant information and/or services” [7] to users or, by extension, to other Agents.

We present an architecture specifically designed to meet the challenges of open intelligent space: it applies a micro Context-awareness approach for Agents that are distributed, autonomic, fully mobile and do not depend on specialized components. Our Agents reason on Context through a local ontology and offer continuity of service even in relative isolation.

This paper is structured as follows: §2 discusses related work; §3 describes our architecture and its main components (§3.2-3.5), its Context model (§3.6) and its programming model (§3.7); §4 concludes with an indication of the work ahead.

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas.
doi:10.1016/j.procs.2011.07.039

2. Related work

There has been a recent surge of interest in decentralized, Context-aware Component (Agent)-oriented middleware. Such middleware typically includes a hierarchical infrastructure of layered or peer-to-peer architectures [8,9] with at least one specialized, continuously available component [10]. The authors of [11] use distributed agents with local plans that communicate based on the Belief-Desire-Intention model, and use specialized components such as a policy server and a context information server. Harroud et al. [12] use policy-based reasoning for Agents with classic Context categories (time, space, location, identity, and a service-to-user mapping). Kovacs et al. [13] define a mobile agent model with migratory capabilities over a wireless network based on a single binary format but with a Context model strongly different from ours. Developed at DOMUS laboratory [14], our architecture targets open intelligent space: it relies on no specialized node, and uses Agent-local Context definitions and ontology.

Context-awareness is used in situations where software and hardware have to cope with complex data. It often involves semantic Web technology [15] or devices that use external data to build a model (i.e. an awareness) of their surroundings, relationships with other devices [16] or subject of interest [17]. Context-awareness also applies to software that assists users and takes into account the relationships between users and their environment.

Context can refer to the situation of a device or of a human being. In [6], Context is defined as "a setting in which an event occurs", which ties Context to a location. For the authors of [18], "context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". In [19], a distinction is made between situation-awareness, related to the user's location, and Context-awareness, related to conditions (temperature, weather, lighting, …) that prevail in that location.

Context-awareness is made difficult by the complexity of augmented reality. It may involve many aspects such as satisfying real-time constraints, making statistical inferences [20] to build synthetic information from raw data or other synthetic information, filtering information [19], specifying agents [21] to distribute subsets of the required processing over many processing nodes, building service publishing and discovery mechanisms [22], using distributed data sources [23] or managing conflicting data and faulty components [24].

Context-aware system design is guided by questions such as "what is aware?", "what is it aware of?", "what actions have to be taken given this awareness?" and "for how long does this awareness remain valid?" [25]. Awareness can be seen as an emergent property of a complex system of Context-dependent software Agents. These systems assist people with disabilities performing activities of daily life; ensure maximal comfort of tourists in hotel resorts by adapting their environment to their needs and preferences; control the way people move between given areas [26] and automatically adapt car seat temperature depending on the individual seated [27].

To achieve autonomic pervasive computing, Context-aware Agents emphasize the independence of each device in its Context reasoning [2], using orchestration of inner devices' knowledge to dispatch processes in pervasive environments [28], or using global environment knowledge of Context to provide applications to users' devices within controlled smart spaces [29,30]. Solutions exist to specific subsets of open space problems, such as ubiquitous location discovery [31,32], but the general open space problem remains open.

3. Architecture

An overview of the overall platform architecture used in our group is provided in this section. The main components of the architecture are described and put in relation to one another. A short discussion of Context as we define and model it is given in a later section.

3.1. Overview

We designed a distributed multi-level architecture ($L_0..L_n$) that integrates the micro and macro Context-awareness models by providing structures, Context descriptions, ontologies and services to embedded middleware and software.

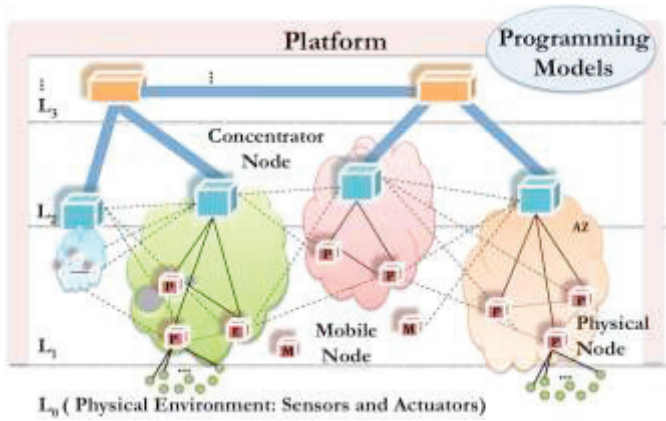

Fig. 1. Overview of the overall architecture used for Open smart space development

Figure 1 presents this multi-level architecture from a macro perspective. From this perspective, the architecture is instrumented with customized nodes, fixed or mobile. Each L_i node offers L_i-level services. L_1 nodes are hardware abstraction gateways that enable the system to interact with L_0 real world data (sensors and actuators). $L_i, i > 1$ nodes are concentrators that aggregate the services offered by $L_j, j < i$ nodes in a zone (location of influence). These concentrator nodes aggregate lower-level services in a given zone. Zones group nodes according to some criteria (for example, physical proximity), and can overlap. The hierarchy can be based on Agent needs: location (e.g. L_1 is for devices, L_2 for a room, L_3 for a floor…), processing requirements (e.g. a $L_i, i > 1$ node concentrates instances of L_{i-1} nodes, to perform load balancing), or to address security, confidentiality and ethical concerns. Mobile nodes can, at any time, gain or lose contact with individual nodes due to faults, distance, noise or other factors.

In hierarchical scenarios, L_0 devices gather information from sensors to build Context. This Context is retrieved by L_1 devices and is used to create the first Context-aware functionalities related to a component and its neighborhood, i.e. raw Context. L_2 devices aggregate micro Context information from L_1 devices, building a larger vision of Context for a given zone, leading to Context-awareness from a macro perspective. In layers L_3 and up, devices aggregate Context from lower layers and construct a global vision of the known environment. Mobile micro Context-aware Agents, on the other hand, move freely from zone to zone. Both their network neighborhood and their conceptual level change with time and location. The application perspective of macro Context-aware systems, where Agents have well-known roles in a well defined system, is replaced by localized, per-Agent roles, resulting in a local – rather than systemic – perspective.

To support this micro approach, our architecture includes the following key elements: Host components, Agents, Context and Context space, both for individual Agents and for Host components themselves. We describe each of these elements in more detail in the following sections.

3.2. Host components

We apply a model where sets of Agents on individual fixed or mobile nodes $L_i i \geq 0$ are managed as smaller systems in special components named Host components, or simply Hosts. Thus, Hosts are required to perform continuous network discovery and interact with various other nodes. The network neighborhood of Host H at time t is the set of Hosts with which H can interact at that time. Consequently, the level of a given Host component in the hierarchy described in figure 1 varies over time. A high-level view of a Host component is shown in figure 2.

Our Host components are similar to OSGi bundles [34] and OpenCom components [35]; they manage deployment and execution of Agents, and trade exclusively in Context. Hosts ensure that Agents (§3.3) work strictly on local knowledge found in their Context space (§3.5).

Context and services are consumed and exposed by individual Agents through their Host. Hosts serve as middleware; they interact with one another and with nodes based on other technologies on behalf of their hosted Agents, and they manage Agents through a specific life cycle. Hosts responsibilities include network isolation, neighborhood discovery and overall management of the Agents' life cycle (§3.4).

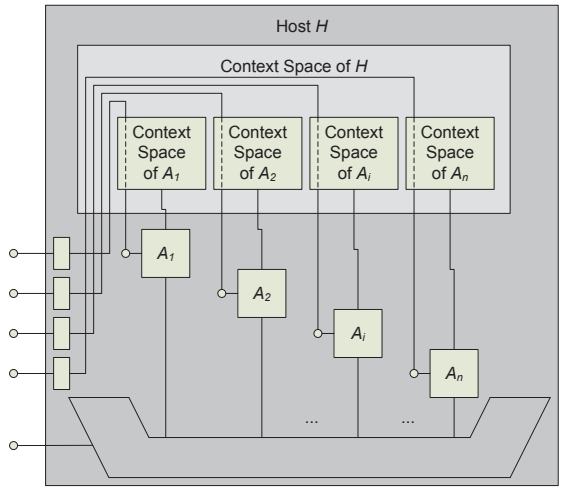

Fig. 2. High-level view of a Host component

To support full mobility of Agents between zones, the representation of a node's network neighborhood dynamically changes in the Host's Context space. The logical neighborhood represented in Context space at time t is not necessarily isomorphic with the actual network topology in which the Host is involved; Agents only operate on the logical, Context-based counterpart of the network and remain isolated from these concerns.

Through network isolation, Hosts recognize one another, build ad hoc network neighborhoods, and communicate directly with one another. This lets Agents operate locally, free from networking and concurrency difficulties. Communication between Agents is mediated by the Context space of every Host involved.

3.3. Agent model

All Agents in our architecture are Context-dependent, and are involved in a continuous Context-dependency management cycle as described by figure 3. Context is the key format used to represent augmented reality throughout our architecture.

We distinguish between standard Agents and domain-specific (or user) Agents; this distinction is reflected in our life cycle management (§3.4). Standard Agents perform tasks for their Host. They implement the Host's services, and as such, they have privileges user Agents do not have. Standard Agents serve as infrastructure services,

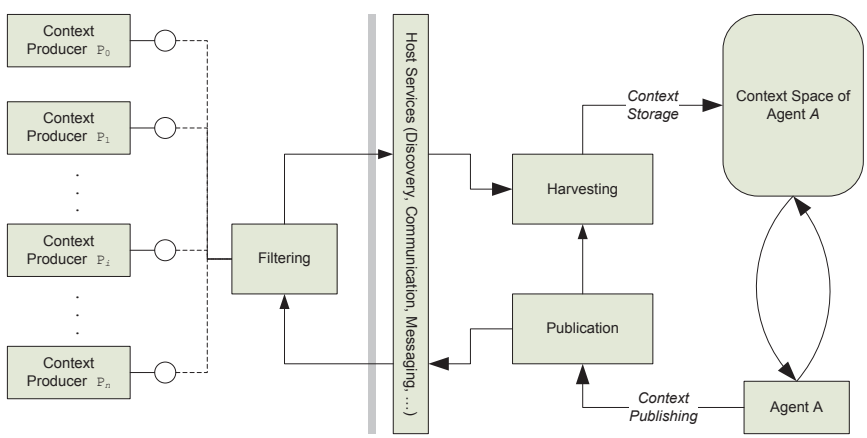

Fig. 3. Overview of Agents' Context-dependency management cycle

supporting the Host and keeping Host software proper as minimal and lightweight as possible. User Agents, or simply Agents, perform Context-aware tasks to solve specific problems. In our architecture, these Agents play the role that applications serve in macro Context-aware systems.

Hosts isolate Agents from one another completely, preventing direct interaction both between different Hosts and within a single Host, since all interaction is mediated through Context space (§3.5).

Agents trade Context through their respective Hosts and typically express themselves through a Context variant of the Belief-Desire-Intention model [11], due to the difficulty of actually establishing facts about augmented reality in the presence of conflicting sources of information and potentially faulty components.

3.4. Life cycle management

An Agent is deployed on a Host, and each Host serves as an executive for all Agents that have been deployed on it. A Host's life cycle is oriented toward the management of hosted Agents. As shown in figure 4, during a Host's execution, each Agent is:

- loaded into the Host process' node's memory;
- activated;
- used, (allowed to perform its tasks);
- passivated; and
- persisted.

The activation and passivation steps for an Agent can occur many times between the moment when this Agent is loaded into memory and the moment it is persisted. Standard Agents are loaded and activated before user Agents, just as they are passivated and persisted after user Agents. The reason for this is that standard Agents are part of the basic set of Host services and can be expected to be available for user Agents throughout their execution.

Since our Agents are Context-dependent, they run through a smaller cycle composed of the following steps: consuming Context, processing Context, and publishing Context. Each of these steps is optional and can be empty for a given Agent. For example, a Context-relaying Agent that extends the range of other Context producers could consume and publish without performing any processing; an Agent on an actuator node could be a strict Context consumer, while an Agent on a sensor node could be a strict Context producer. Standard Agents either execute before or after user Agents, depending on their tasks.

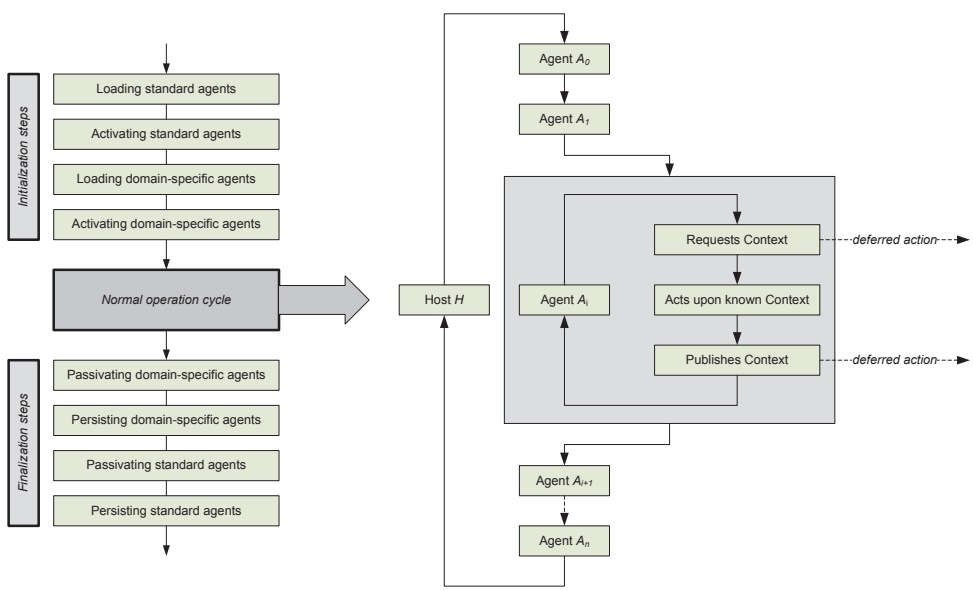

Fig. 4. (a) Host life cycle (general), (b) closer look at the normal Agent operation cycle

The normal operation cycle of Agent $A_i, 0 \leq i \leq n$ in Host H, when taken generally, is shown in figure 4. This figure provides a high-level description of a Context-requesting, gathering, processing and producing cycle.

Suppose A_i seeks to evaluate a relative comfort index (a *synthetic* Context) inferred from two lower-level (maybe *raw*) Contexts such as ambient temperature and humidity. The requesting phase can involve A_i using Context to describe requests for a Context description of individual temperature and humidity; the publishing phase can involve A_i using Context to describe the most recent synthetic comfort value it has computed, and the action (processing) phase can involve the actual computation based on the current state known to A_i for each Context relevant to that computation. A_i always operates on the current state of its knowledge as described in its Context space. There is no assumption that the current state of the knowledge at time t is exact with respect to conditions prevailing in the physical world at time t; A_i simply acts on the most appropriate Context to which it has access.

The requesting and publishing phases in figure 4 are marked as *deferred actions*. Deferred actions are part of our programming model (§3.7) and are asynchronous actions started by the requesting Agent, and managed by the Host component in such a way that they can eventually come to affect the contents of the Agent's Context space.

3.5. Context space

We name Context space an organized Context repository, where all Context on Host H is stored in the Context space of H; similarly, for any Agent A, the Context known to A is stored in A's Context space.

On any Host H, the Context space of H subsumes the Context spaces of all Agents deployed on H. For Agents, individual Contexts in a given Context space are immutable, which allows Hosts to share Context between Agents. Thus, consumption and production of Context for Agent A appear to be synchronized in the Context space of A.

Requests for Context are described in the form of Context by the requesting Agent; they are identified with the requesting Agent and mediated through Context space. The Host, through standard Agents, consumes requests and turns them into searches within the local Context space and into neighboring Hosts' Context spaces. An Agent operates on local versions of Context that its Host has obtained in response to its requests.

A Host's Context space contributes to its network isolation by holding a representation of Context that hosted Agents have either published or requested and obtained, as well as what standard Agents have provided: this includes such information as a Context representation of the underlying node's properties and a Context representation of the actual network neighborhood of the Host.

3.6. Overview of the Context model

We define Context as a recursive key-value structure where the key is a unique identifier and the value is a potentially empty set of Context [3]. Our Context-aware components represent as Context the traditional [12] categories of Context (Activity, Identify, Location, Time), as well as the description of published Context, requests for Context given specific constraints and the criteria for filtering Context in order to obtain an ontologically meaningful subset for a given Agent. Context is managed through Context space (§3.5).

Since we use Context for all augmented reality-related information, we include as Context self-descriptive data and metadata such as individual node status and Agent services. In so doing, Agents can perform their tasks in a completely Context-dependent and Context-aware manner.

In our architecture, every Agent has a local ontology. This impacts the contents and management of Context space: the requirements and productions of hosted Agents have a direct influence on the contents of Context space, and thus on what actually constitutes Context on a given Host. A consequence of this design choice is that we use a *Context-according-to* model, with emphasis put on matching requests for Context with published Context. Instead of sharing a common ontology for all Agents, we use a common format to express published and requested Context and automate the matching process between produced and requested Context using a single engine, reducing the complexity of Host software. User Agents only have to implement the functions required to consume, process and publish Context in order to perform their tasks.

We use a mix of generic algorithms and criteria to operate on Context from a local, Agent-based perspective [3]. This approach has been successful in industrial software development [36] and lets us build powerful abstractions from a combination of simpler components.

3.7. Overview of the programming model

Since the Host manages the execution of individual Agents, actual modifications to a Host's Context space can be performed at "safe" moments, simplifying concurrency management. To keep Context space concurrency-free and to ensure Agents can be deployed even on nodes with limited resources, we impose restrictions on user Agents: they cannot spawn threads and should perform linear tasks when consuming, processing or producing Context. The Host, through its life cycle, offers individual Agents regular processing opportunities. Host implementations themselves can be multithreaded if the underlying operating system allows it.

Operations that require concurrency are under the responsibility of the Host itself. Network neighborhood discovery and update is an example of such a task, and is performed in parallel to the normal life cycle of the Host. Synchronized data structures are used between these concurrent tasks and Context space; standard Agents are responsible for accessing these synchronized objects and managing integration between their contents and the Context space within the normal, concurrency-free operation cycle.

4. Conclusion and future works

To approach Context-awareness problems in open intelligent space, we apply a micro approach using distributed Agents, each with its own local ontology. Agents are deployed on Host components, which are in turn deployed on physical nodes. We represent augmented reality with Context, mediated through the Context space of individual Agents and of their Host.

The models are currently being refined and the following items are being investigated:

- automatically transforming Context requests into operations is a promising but ongoing process. Refining the grammar for Context-formatted patterns will allow more complex requests on the part of Context consumers;
- the management of Context space, for reasons such as identifying Context that has become stale, is a problem that has to be addressed in order to manage resources efficiently;
- different avenues have to be explored in order to facilitate the deployment and migration of Agents, including representing Agents as Context and migrating binaries to compatible Hosts;
- the algorithmic complexity of Context management has to be studied in order to improve such characteristics of autonomic computing as self-optimization of Agents.

Our architecture in its current state is promising. The addition of Agents has proven to be a seamless process, and the network isolation through Context space has proven to be very helpful in building stable components for unstable environments.

References

1. Roy, P., Abdulrazak, B. and Belala, Y., Approaching Context-awareness for Open Intelligent Space, Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia (2008) 422-426.

2. Kephart J. O. and Chess D. M., The vision of autonomic computing, IEEE Computer, 36 (2003) 41-50.

3. Abdulrazak, B. *et al*, Micro Context-Awareness for Autonomic Pervasive Computing, accepted for publication in the International Journal of Business Data Communications and Networking (IJBDCN), (2011).

4. Weiser, M., The computer for the 21st century, Scientific American (1991) 94-104.

5. Winograd, T., Architectures for Context, Human-Computer Interaction, 16(2) (2001) 401-419.

6. Neovius, M. *et al*, A Formal Model of Context-Awareness and Context-Dependency, Software Engineering and Formal Methods (2006) 177-185.

7. Fujii, A., Trends and Issues in Research on Context Awareness Technologies for a Ubiquitous Network Society No. 16 (2008) 26-35.

8. GAIA project http://gaia.cs.uiuc.edu

9. Salber, D., *et al.*, The context toolkit: aiding the development of context-enabled applications, ACM-SIGCHI conf. on Human factors in computing systems. Pittsburgh, USA (1999) 434-441.

10. Sahil, N., Survey: Agent-based Middlewares for Context Awareness, Electronic Communications of the EASST, vol. 11 (2008)

11. Plesa, R. and Logrippo, L., An Agent-Based Architecture for Context-Aware Communication, 21st IEEE Conf. on Advanced Information Networking and Applications, Niagara Falls, ON, Canada (2007) 133-138.

12. Harroud, H. and Karmouch, A., A Policy Based Context-Aware Agent Framework to Support Users Mobility, IEEE AICT-SAPIR-ELETE (2005) 177-182.

13. Kovacs, E., Röhrle, K. and Schiemann, B., Adaptive Mobile Access to Context-Aware Services, 1st IEEE Int. Symposium on Agent Systems and Applications, Palm Springs, CA, USA (1999) 190-201.

14. DOMUS Laboratory, http://domus.usherbrooke.ca/

15. Feki, M. A and Mokhtari M., Context aware and ontology specification for assistive environment, HWRS-ERS Journal, International Journal of Human-friendly Welfare Robotic Systems, 4(2) (2006) 29-32.

16. McCann J.A., Kristofferson P. and Alonso, E., Building Ambient Intelligence into a Ubiquitous Computing Management System, International Symposium on Challenges in the Internet and Interdisciplinary Research (2004).

17. Miaou, S.-G., Shih, F.-C. and Huang, C.-Y., A Smart Vision-Based Human Fall Detection System for Telehealth Applications, IASTED Telehealth Conference (2007) Montreal, QC, Canada.

18. Abowd, G. D. *et al*, Towards a Better Understanding of Context and Context-Awareness, Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing (1999), Karlsruhe, Germany.

19. Gessler, S.; Martin, M.; and Weiss, S., Context Awareness in Future Life Scenarios: Impact on Service Provisioning Platforms, Applications and the Internet Workshops (2005) 144–147.

20. Katsiri, E., Principles of Context Inferences, International Conference on Ubiquitous Computing (2002), Gotenborg, Sweden.

21. Zaslavsky, A., Mobile Agents: Can They Assist with Context Awareness?, Proceedings of the 2004 IEEE International Conference on Mobile Data Management (2004).

22. Gouin-Vallerand, C. and Giroux, S., Managing and Deployment of Applications with OSGi in the Context of Smart Home, Proceedings of the Third IEEE international Conference on Wireless and Mobile Computing, Networking and Communications (2007).

23. Kiani, S. L. *et al*, A Distributed Middleware Solution for Context Awareness in Ubiquitous Systems, Proceedings of the 11th IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications (2005).

24. Agostini, A., Bettini, C., and Riboni, D., Loosely Coupling Ontological Reasoning with an Efficient Middleware for Context-awareness, Proceedings of the Second Annual international Conference on Mobile and Ubiquitous Systems: Networking and Services (2005).

25. Kernchen, R. *et al*, Context-Awareness in MobiLife, Proceedings of the 15th IST Mobile Summit, Mykonos, Greece (2006).

26. Sun, J. and Song Dong, J., Design Synthesis from Interaction and State-Based Specifications, IEEE Trans. Softw. Eng. 32(6), (2006) 349-364.

27. Yaiz, R. A., Selgert, F. and den Hartog, F., On the definition and relevance of context-awareness in Personal Networks, 3rd Annual International Conference on Mobile and Ubiquitous Systems (2006) 1-6.

28. Trumler W., Klaus R. and Ungerer T., Self-configuration via cooperative social behavior, ATC; Lecture Notes in Computer Science (2006) 90-99.

29. Ranganathan A., Shankar C. and Campbell R., Application polymorphism for autonomic ubiquitous computing, Multiagent Grid Syst., No. 1 (2005) 109-129.

30. Ghorbel M., Mokhtari M. and Renouard S., A distributed approach for assistive service provision in pervasive environment, Proceedings of the 4th International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (2006) 91-100.

31. Abdulrazak, B., Chakroun, O., Malik, Y., LocSys Localization framework for smart spaces, Gerontechnology 2010, (2010) Vancouver, Canada.

32. Chakroun, O. *et al*, Indoor and Outdoor Localization Architecture for Pervasive Environment, 8th International Conference on Smart Homes and Health Telematics (ICOST), Seoul, Korea (2010) 242-245.

33. Truong, H.-L. & Dustdar, S., A Survey on Context-aware Web Service Systems, International Journal of Web Information Systems, 5(1) (2009) 5-31.

34. OSGi Alliance (2009). OSGi Technology, retrieved April 18, 2011 from http://www.osgi.org/About/Technology.

35. Coulson, G. *et al*, A Generic Component Model for Building Systems Software, ACM Transactions on Computer Systems, 26(1) (2008) 1-42.

36. Stepanov, A. and McJones, P., Elements of Programming, Boston, MA, USA: Addison-Wesley Professional (2009).